CYPR-CD01208M

UNITED STATES PATENT APPLICATION FOR

SLEEP AND STALL IN AN IN-CIRCUIT EMULATION SYSTEM

Inventor:

Craig Nemecek

Prepared By:

Wagner, Murabito & Hao LLP
Two North Market Street
Third Floor
San Jose, CA 95113
Phone: (408) 938-9060
Fax: (408) 938-9069

SLEEP AND STALL IN AN IN-CIRCUIT EMULATION SYSTEM

FIELD OF THE INVENTION

This invention relates generally to the field of In Circuit Emulation. More particularly, this invention relates to methods and apparatus for performing sleep and stall operations in an in-circuit emulation system.

BACKGROUND OF THE INVENTION

In-circuit emulation (ICE) has been used by software and hardware developers for a number of years as a development tool to emulate the operation of complex circuit building blocks and permit diagnosis and debugging of hardware and software. In-circuit emulation is most commonly used to analyze and debug the behavior of complex devices such as microcontrollers and microprocessors.

In conventional in-circuit emulation systems, a host computer (e.g., a personal computer) is connected to a debug logic block which is further connected to a special version of the microcontroller device that has been developed specially for use in emulation. Some in-circuit emulation systems use a bond-out version of the microcontroller that includes wirebonding pads on the chip that are not normally connected in the production wirebonding. These pads are connected to pins on the microcontroller package to permit access to otherwise inaccessible points of the circuit to facilitate debugging. This technique

has the disadvantage of imposing significant limitations on the circuit layout to permit space and circuitry associated with the special wirebonding pads. Also, interface circuitry and other special circuitry is typically added to facilitate the debugging and bond-out. This increases the complexity, size, power consumption and potentially reduces the yield of the production part. Moreover, development resources are required to design and lay out the bond-out circuitry and pads.

Other in-circuit emulation systems use a special "probe mode" of operation of the processor in which a number of internal signals are routed to a "debug port" for use by the in-circuit emulation system. In these systems, the debug port allows the in-circuit emulation system to communicate with the processors at all times and, when placed in probe mode, to read otherwise inaccessible probe points within the processor. Of course, providing such a probe mode requires significant design resources, increasing development cost, chip complexity and chip size.

In in-circuit emulation operations, operational instructions are loaded from the host computer through the debug logic to the special version of the microcontroller. The debug logic monitors operation of the microcontroller as the instructions are executed. Depending upon the application, this operation may be monitored while the special version of the microcontroller is interconnected with the circuitry that is intended to interface a production version of the

microcontroller in the finished product under development. As the circuit is stepped through its operation, the debug logic gathers information about the state of various components of the microcontroller during operation and feeds that information back to the host computer for analysis.

During the course of the analysis, various trace information such as time stamps, register values, data memory content, etc. may be logged in the host computer for analysis and debugging by the designer. Additionally, it is generally the case that various break points can be defined by the designer that cause the program to halt execution at various points in the operation to permit detailed analysis. Other debugging tools may also be provided to enable the user to debug the operation of the circuit.

During the course of the analysis, the microcontroller can perform a sleep function in which the microcontroller "sleeps" or pauses some or all operations. During a typical sleep function the CPU clock of the microcontroller stops operating to conserve battery power. Also, the microcontroller can perform a stall operation in which the microcontroller pauses some operations. During a typical stall operation the CPU clock of the microcontroller continues to operate. Sleep and stall operations should be addressed in in-circuit emulation systems to assure that the in-circuit emulation system emulates the sleep or stall operation.

SUMMARY OF THE INVENTION

The method and apparatus of the present invention effectively provides in-circuit emulation using an emulation device that operates in lock-step fashion with the device under test. The method and apparatus of the present invention effectively handles sleep and stall operations such that the emulation device and the device under test continue to operate in lock-step after sleep and stall operations have been performed.

A system that includes a device under test and that includes an emulator device is disclosed. The emulator device emulates the functions of the device under test by operating in lock-step fashion with the device under test. In one embodiment, the emulator device is a Field Programmable Gate Array (FPGA) device and the device under test is a microcontroller. A host PC can be coupled to the emulator device. The FPGA is programmed to operate as a virtual microcontroller, performing a set of instructions that are also performed by the microcontroller in lock-step fashion. In the present embodiment, only the core processing functions of the microcontroller are performed in lock-step fashion. However, alternatively, any or all instructions or sequences of instructions performed by the microcontroller could be performed in lock-step fashion by the FPGA.

More specifically, for a system that includes a device under test and that includes an emulator device that emulates the functions of the device under test

by operating in lock-step fashion with the device under test, a method for performing a sleep operation is disclosed in which the device under test initiates the sleep function upon receiving a first signal that indicates that a sleep function is to be performed and turns off its clocks. In one embodiment, the first signal is generated internally by the device under test and is transmitted internally to a register that indicates that a sleep function is to be performed.

The emulator device discontinues execution of the sequence of instructions that are performed in lock-step when the clock is turned off. More particularly, when clock signals are no longer received at the emulator device, the emulator device discontinues execution of the sequence of instructions that are performed in lock-step.

When the sleep function has been completed by the device under test a second signal (the wake-up signal) is sent from the device under test to the emulator device. Upon receiving the second signal at the emulator device, the emulator device determines the number of clock signals received at the emulator device since the second signal was received. Execution of the sequence of instructions that are performed in lock-step fashion at the emulator device is resumed when the determined number of clock signals received equals a predetermined value.

Also, for a system that includes a device under test and that includes an emulator device that emulates the functions of the device under test by operating in lock-step fashion with the device under test, a method for performing a stall operation is disclosed. In the present embodiment, the device under test conveys clock signals (e.g., from the microcontroller CPU clock) to the emulator device.

The device under test initiates the stall function upon receiving a first signal that indicates that a stall function is to be performed. In one embodiment, the first signal is generated internally by the device under test and is transmitted internally to a register that indicates that a stall function is to be performed. Upon receiving the first signal, the device under test discontinues sending clock signals to the emulator device. The emulator device discontinues execution of instructions that are performed in lock-step fashion while sending of clock signals is discontinued.

When the stall function has been completed by the device under test sending of clock signals to the emulator device is resumed. Upon receiving the clock signals, the emulator device resumes execution of the sequence of instructions that are performed in lock-step.

Accordingly, both the device under test and the emulator device stop and resume execution of the sequence of instructions that are performed in lock-step

fashion during both sleep and stall functions in such a manner as to assure that the device under test and the emulator device remain in lock-step.

Thereby, the method and apparatus of the present invention effectively provides in-circuit emulation using an emulation device that operates in lock-step with the device under test. Also, the method and apparatus of the present invention effectively handles sleep and stall operations such that the emulation device and the device under test continue to operate in lock-step after sleep and stall operations are completed.

It is appreciated that the emulator and emulation methods described herein can work equally well with any programmable device, and that the microcontroller described herein is one example.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention.

FIGURE 1 is a block diagram of an exemplary In-Circuit Emulation system in accordance with an embodiment of the present invention.

FIGURE 2 is a block diagram that shows the host to FPGA interface in accordance with an embodiment of the present invention

FIGURE 3 is an illustration of the operational phases of an In-Circuit Emulation system in accordance with an embodiment of the present invention.

FIGURE 4 is an illustration of the operational phases of an In-Circuit Emulation system viewed from a virtual microcontroller perspective in accordance with an embodiment of the present invention.

FIGURE 5 is an exemplary timing diagram illustrating a data and control phase of operation in accordance with an embodiment of the present invention.

FIGURE 6 is a flow chart that illustrates a method for performing a sleep operation in accordance with an embodiment of the present invention.

FIGURE 7 is a flow chart that illustrates a method for performing a stall operation in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be recognized by one skilled in the art that the present invention may be practiced without these specific details or with equivalents thereof. In other instances, well-known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

NOTATION AND NOMENCLATURE

Some portions of the detailed descriptions which follow are presented in terms of procedures, steps, logic blocks, processing, and other symbolic representations of operations on data bits that can be performed on computer memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, computer executed step, logic block, process, etc., is here, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities.

Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It has proven convenient at

times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "generating" or "executing" or "determining" or "conveying" or "initiating" or "sending" or "receiving" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

While this invention is susceptible of embodiment in many different forms, there is shown in the drawings and will herein be described in detail specific embodiments, with the understanding that the present disclosure is to be considered as an example of the principles of the invention and not intended to limit the invention to the specific embodiments shown and described. In the description below, like reference numerals are used to describe the same, similar or corresponding parts in the several views of the drawings.

IN-CIRCUIT EMULATION SYSTEM

Referring now to FIGURE 1, a system 200 is shown that includes a host computer 210 (e.g., a personal computer based on a Pentium™ class microprocessor) that is interconnected (e.g., using a standard PC interface 214 such as a parallel printer port connection, a universal serial port (USB) connection, etc.) with a base station 218. The host computer 210 generally operates to run an ICE computer program to control the emulation process and further operates in the capacity of a logic analyzer to permit a user to view information provided from the base station 218 for use in analyzing and debugging a system under test or development.

The base station 218 is based upon a general-purpose programmable hardware device such as a gate array configured to function as a functionally equivalent "virtual microcontroller" 220 (or other device under test (DUT)). This is accomplished using an associated integral memory 222 which stores program instructions, data, trace information and other associated information. Thus, the base station is configured as an emulator of the internal microprocessor portion of the microcontroller 232. In preferred embodiments, a field programmable gate array FPGA (or other programmable logic device) is configured to function as the virtual microcontroller 220. More particularly, virtual microcontroller 220 is implemented within the FPGA of base station 218. The base station 218 is coupled (e.g., using a four wire interface 226) to a standard production

microcontroller 232 mounted in a mounting device referred to as a "pod". The pod, in certain embodiments, provides connections to the microcontroller 232 that permit external probing as well as interconnection with other circuitry as might be used to simulate a system under development.

In one embodiment, system 200 is adapted to test the CY8C25xxx/26xxx series of microcontrollers made by Cypress Micro Systems, Inc., 22027 17th Avenue SE, Suite 201, Bothell, WA 98021Bothell, WA. In this embodiment FPGA 220 emulates the core processor functionality (microprocessor functions, Arithmetic Logic Unit functions and RAM and ROM memory functions) of the Cypress CY8C25xxx/26xxx series microcontrollers. Detailed information regarding this commercial product is available from Cypress Micro Systems, Inc., in the form of version 1.11 of "PSoC Designer: Integrated Development Environment User Guide", which is hereby incorporated by reference in its entirety as background material. While the present invention is described in terms of an ICE system for the above exemplary microcontroller device, the invention is equally applicable to other complex circuitry including microprocessors and other circuitry that is suitable for analysis and debugging using in-circuit emulation. Moreover, the invention is not limited to the exact implementation details of the exemplary embodiment used herein for illustrative purposes.

In order to minimize the need for any special ICE related functions on microcontroller 232, the FPGA 220 and associated circuitry of the base station 218 are designed to operate functionally in a manner identically to that of the microprocessor portion of the production microcontroller, but to provide for access to extensive debug tools including readout of registers and memory locations to facilitate traces and other debugging operations.

Virtual microcontroller 220 operates to execute the code programmed into microcontroller 232 in lock-step operation with microcontroller 232. Thus, there is no need to provide special facilities for ICE in microcontroller 232, since any such facilities are provided in virtual microcontroller 220. Virtual microcontroller 220 and microcontroller 232 operate together such that I/O reads and interrupts are fully supported in real time. The combination of real and virtual microcontroller behave just as the microcontroller 232 would alone under normal operating conditions. I/O reads and interrupt vectors are transferred from the microcontroller 232 to the base station 218 as will be described later. Base station 218 is then able to provide the host computer 210 with the I/O reads and interrupt vectors as well as an array of information internal to microcontroller 232 within memory and register locations that are otherwise inaccessible.

In the present embodiment, the design of microcontroller 232 is implemented using the Verilog™ language (or other suitable language). Thus, the full functional design description of the microcontroller is available in a

software format. In one embodiment base station 218 is based upon the commercially available Spartan™ series of FPGAs from Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. The Verilog™ description can be used as the input to the FPGA design and synthesis tools available from the FPGA manufacturer to realize virtual microcontroller 220 (generally after timing adjustments and other debugging). Thus, design and realization of the FPGA implementation of an emulator for the microcontroller (virtual microcontroller) or other device can be readily achieved by use of the Verilog™ description along with circuitry to provide interfacing to the base station and the device under test (DUT).

Continuing with FIGURE 1, the actual production microcontroller 232 carries out its normal functions in the intended application and passes I/O information and other information needed for debugging to FPGA 220. Virtual microcontroller 220 serves to provide the operator with visibility into the core processor functions that are inaccessible in the production microcontroller 232. Thus, FPGA 220, by virtue of operating in lock-step operation with the microcontroller 232 provides an exact duplicate of internal registers, memory contents, interrupt vectors and other useful debug information. Additionally, memory 222 can be used to store information useful in trace operations that is gathered by the FPGA 220 during execution of the program under test. This architecture, therefore, permits the operator to have visibility into the inner workings of the microcontroller 232 without need to provide special bondouts and expensive circuitry on the microcontroller itself.

Virtual microcontroller 220, operating under control of host computer 210, carries out the core processor functions of microcontroller 232 and thus should always try to contain a functionally exact emulated copy of the contents of the registers and memory of microcontroller 232. To maintain this data integrity, certain data is passed from the microcontroler to the FPGA, e.g., data that comes on an external data or peripheral bus coupled only to the microcontroller. The ICE system starts both microcontrollers (real and virtual) at the same time and keeps them running in synchronization. Microcontroller 232 sends I/O data to base station 218 (and in turn to the ICE software operating on the host computer 210 if required) fast enough to keep the microcontroller 232 and virtual microcontroller 220 of base station 218 in synchronization. Whenever the system is halted (i.e., when the system is not emulating), other information (e.g., flash memory programming functions, test functions, etc.) can be sent over the interface.

Because microcontroller 232 operates in synchronization with virtual microcontroller 220, less data needs to be sent over the four-wire interface than would be required in an ICE system otherwise. The type of data sent over the lines is allowed to change depending on when the data is sent in the execution sequence. For example, depending on the execution sequence time, the information sent over the data lines can be commands to microcontroller 232 or can be data. Since the clock frequency of microcontroller 232 is programmable,

it copies its current clock on one of the lines of the four wire interface. Moreover, virtual microcontroller 220 does not require certain resources of the microcontroller 232 such as timers, counters, amplifiers, etc. since they are fully implemented in virtual microcontroller 220. In addition, since all registers and memory locations, etc. are available through the virtual microcontroller 220, the microcontroller 232 (or other DUT) can be debugged in real time without need for extensive debug logic residing on the microcontroller 232.

In the embodiment illustrated, the basic interface used is a four-line interface between microcontroller 232 and base station 218. The four-wire interface 226 of the present embodiment can be functionally divided into two functional portions, a data transport portion 242 and a clock portion 246. Three additional lines are also provided (not shown) for supply, ground and a reset line.

The data transport portion 242 includes two data lines. The first data line (data1) provides a data signal to send I/O data to virtual microcontroller 220. The first data line is also used to notify FPGA 220 of pending interrupts. The Data1 line is only driven by microcontroller 232. A second data line (Data2), which is bidirectional, is used by the microcontroller 232 to send I/O data to the FPGA based virtual microcontroller of base station 218. In addition, the FPGA 220 uses the Data2 line to convey halt requests (e.g., to implement simple or complex breakpoints) and other information to the microcontroller 232.

The clock portion 246 includes a debug system clock (data clock) line and a microcontroller clock line. The data clock line provides a data clock signal (U_HCLK) from a 24/48MHz data clock driven by microcontroller 232. This clock is used to drive the ICE virtual microcontroller communication state machines (the logic used within the state controller to communicate with the microcontroller 232) and to regulate data transfer and other operations. The second clock interface line (uCONTROLLER CLOCK) is the internal microcontroller CPU clock of microcontroller 232. In the present embodiment, the data clock runs at 24MHz, unless the internal microcontroller 232 clock is running at 24MHz (when the clock (U_CCLK) of microcontroller 232 is running at 24MHz, the data clock signal switches to 48MHz).

In the present embodiment, the four-line interface 226 forms a part of a seven wire connection as described below. The interface signals travel over a short (e.g., one foot) of CAT5 network cable. In the present embodiment, a fifth line (not shown) is be used to provide a system reset signal to effect the simultaneous startup of both microcontrollers. This fifth line provides a convenient mechanism to reset the microcontrollers, but in most environments, the simultaneous startup can also be effected in other ways including switching of power. In the present embodiment, the reset signal line outputs an optional active high reset signal (ICE_POD_RST) to microcontroller 232. Sixth and Seventh lines (not shown) are provided in the current interface to provide power and ground for power supply. More particularly, an optional power supply wire

provides power (ICE_POD_PW_R) to microcontroller 232 and an optional ground wire provides ground (ICE_POD_GND) to microcontroller 232.

Synchronization between microcontroller 232 and virtual microcontroller 220 is achieved by virtue of their virtually identical operation. They are both started simultaneously by a power on or reset signal. They then track each other's operation continuously executing the same instructions using the same clocking signals. The system clock signal and the microcontroller clock signal are shared between the two microcontrollers (real and virtual) so that even if the microprocessor clock is changed during operation, they remain in lock-step.

Referring now to the block diagram of FIGURE 2, the interface between the host processor 210 and the base station 218 of a preferred embodiment of the present invention is illustrated. In this embodiment, the connection between the host processor 210 and the FPGA 220 is advantageously provided using a standard IEEE 1284 parallel printer cable 214 with communication carried out using a modification of standard EPP (enhanced parallel port) communication protocol. Of particular interest in this communication interface is the data strobe connection 412, the INIT (initialize) connection 416 and the eight data connections (data line 0 through data line 7) 420. These connections are directly connected to the FPGA with the INIT connection connected to the FPGA RESET pin. The data strobe line 412 is connected to the FPGA configuration clock input and the eight data lines 420 are connected to data input pins of the FPGA.

When the software on the host is started, the INIT connection 416 is driven by the host computer 210 to a logic low causing the FPGA to clear its configuration memory 424 and begin receiving configuration data. The configuration data is stored in configuration memory to define the functionality of the FPGA. This configuration data is clocked in eight bits at a time over the data lines 420 using the data strobe signal as a clock signal. That is, an eight bit word is placed on the interface data lines 420 by host processor 210 followed by toggling the data strobe line to clock the data into the FPGA 220. This unidirectional data transfer from the host computer incorporates a set of design parameters that configure the circuitry of the FPGA 220 to function, in part, as a standard IEEE 1284 EPP interface once the FPGA 220 is programmed and functional. This programming configures the FPGA 220 to have an IEEE 1284 EPP interface with the data lines 420 connected to the FPGA as bidirectional data lines, the configuration clock configured to operate as the IEEE 1284 data clock line connected to data strobe 412 and the INIT line 416 continues to drive the FPGA clear and reset function.

Data transfer continues in this manner until the FPGA 220 is fully programmed by virtue of having received the correct amount of data required by the particular FPGA 220 used in base station 218. Thus, each time the host software is initialized, a data transfer to the FPGA 220 occurs to program the FPGA 220 to function in its capacity of a virtual microcontroller (in this

embodiment). Once programming ceases, the FPGA 220 operates as a virtual microcontroller (or whatever device is programmed into the FPGA 220 in general). At this point, the interface 214 ceases to function as a unidirectional programming interface and begins to function as a bidirectional communication interface using the programmed operation of the FPGA 220 communicating through its programmed IEEE 1248 EPP parallel communication interface.

In the virtual microcontroller mode of operation of the FPGA 220, communication is carried out using the eight data lines 420 as bidirectional data lines compliant with IEEE 1284 EPP parallel communication protocol with the data strobe line 412 used as a data clock and the INIT line 416 continuing to act as a clear and reset signal. INIT line 416 can thus be used to reinitialize the programming of the FPGA 220, for example, to revise a design parameter or to simply restart the ICE system.

In the present embodiment, data strobe interface line provides a unidirectional programming clock in the program mode function and provides an EPP Compliant data strobe in the free running "Awake" mode function. In the program mode function Data bits 0 through 7 provide Unidirectional data into the FPGA while in the free running "awake" mode function data bits 0 through 7 provide EPP compliant communication. A low signal on the INIT interface line in the program mode function indicates clear configuration memory (prepare to receive new configuration data) and low signal on the INIT interface line in the

free running "awake" mode function indicates clear configuration memory and enter programming mode (prepare to receive new configuration data).

Normal operation of the current microcontroller is carried out in a cycle of two distinct stages or phases as illustrated in connection with FIGURE 3. The cycle begins with the initial startup or reset of both the microcontroller 232 and the virtual microcontroller 220 at 304. Once both microcontrollers are started in synchronism, the data phase 310 is entered in which serialized data is sent from the microcontroller to the virtual microcontroller. At the start of this phase the internal start of instruction (SOI) signal signifies the beginning of this phase will commence with the next low to high transition of the system clock. In the current embodiment, this data phase lasts four system clock cycles, but this is only intended to be exemplary and not limiting. The SOI signal further indicates that any I/O data read on the previous instruction is now latched into a register and can be serialized and transmitted to the virtual microcontroller. Upon the start of the data phase 310, any such I/O read data (eight bits of data in the current embodiment) is serialized into two four bit nibbles that are transmitted using the Data0 and Data1 lines of the current interface data portion 242. One bit is transmitted per data line at the clock rate of the system clock. Thus, all eight bits are transmitted in the four clock cycles of the data transfer phase.

At the end of the four clock cycle data transfer phase in the current embodiment, the control phase 318 begins. During this control phase, which in

the current embodiment may be as short as two microcontroller clock periods (or as long as about fourteen clock periods, depending upon the number of cycles required to execute an instruction), the microcontroller 232 can send interrupt requests, interrupt data, and watchdog requests. Additionally, the virtual microcontroller 220 can issue halt (break) commands. If a halt command is issued, it is read by the microcontroller at the next SOI signal. Once the control phase ends, the data transfer phase repeats. If there is no data to transfer, data1 and data2 remain idle (e.g., at a logic low state). To simplify the circuitry, I/O bus data are sent across the interface on every instruction, even if it is not a bus transfer. Since the virtual microcontroller 220 is operating in synchronization with microcontroller 232 and executing the same instructions, the emulation system knows that data transferred during non I/O read transfers can be ignored.

FIGURE 4 shows this operational cycle from the perspective of the virtual microcontroller 220. During the data transfer phase 310, the serialized data is received over Data0 and Data1. It should be noted that prior to receipt of this I/O data, the microcontroller 232 has already had access to this data for several clock cycles and has already taken action on the data. However, until receipt of the I/O read data during the data transfer phase 310, the virtual microcontroller 220 has not had access to the data. Thus, upon receipt of the I/O read data during the data phase 310, the virtual microcontroller 220 begins processing the data to catch up with the existing state of microcontroller 232. Moreover, once the I/O data has been read the host computer 210 or virtual microcontroller 220

may determine that a complex or simple breakpoint has been reached and thus need to issue a break request. Thus, the virtual microcontroller should be able to process the data quickly enough to make such determinations and issue a break request prior to the next SOI. Break requests are read at the internal SOI signal, which also serves as a convenient reference time marker that indicates that I/O data has been read and is available for transmission by the microcontroller 232 to the virtual microcontroller 220.

By operating in the manner described, any breakpoints can be guaranteed to occur in a manner such that both virtual microcontroller 220 and microcontroller 232 halt operation in an identical state. Moreover, although virtual microcontroller 220 and the microcontroller 232 operate on I/O data obtained at different times, both microcontrollers are in complete synchronization by the time each SOI signal occurs. Thus, virtual microcontroller 220 and microcontroller 232 can be said to operate in lock-step with respect to a common time reference of the SOI signal as well as with respect to execution of any particular instruction within a set of instructions being executed by both virtual microcontroller 220 and microcontroller 232.

In accordance with certain embodiments of the invention, a mechanism is provided for allowing FPGA 220 of base station 218 and microcontroller 232 to stop at the same instruction in response to a breakpoint event (a break or halt). The FPGA 220 has the ability monitor the microcontroller states of

microcontroller 232 for a breakpoint event, due to its lock-step operation with microcontroller 232. In the process of executing an instruction, an internal start of instruction cycle (SOI) signal is generated (by both microcontrollers) that indicates that the device is about to execute a next instruction. If a breakpoint signal (a halt or break signal - the terms "halt" and "break" are used synonymously herein) is generated by the FPGA, the execution of the microcontroller 232 can be stopped at the SOI signal point before the next instruction starts.

Although the SOI signal is labeled as a signal indicating the start of an instruction, the SOI signal is used for multiple purposes in the present microcontroller. It is not required that the SOI signal actually indicate a start of instruction for many purposes, merely that there be a convenient time reference on which to base certain actions. For example, any reference signal that always takes place prior to execution of an instruction can be used as a time reference for reading a halt command. Accordingly, any such available or generated reference signal can be used equivalently as a "halt read" signal without departing from the present invention. That not withstanding, the SOI signal is conveniently used in the current embodiment and will be used as a basis for the explanation that follows, but should not be considered limiting.

Logic within the FPGA 220 of base station 218 allows not only for implementation of simple breakpoint events, but also for producing breakpoints

as a result of very complex events. By way of example, and not limitation, a breakpoint can be programmed to occur when a program counter reaches 0x0030, an I/O write is happening and the stack pointer is about to overflow. Other such complex breakpoints can readily be programmed to assist in the process of debugging. Complex breakpoints are allowed, in part, also because the virtual microcontroller 220 has time to carry out complex computations and comparisons after receipt of I/O data transfers from the microcontroller 232 and before the next instruction commences. After the receipt of I/O data from the microcontroller 232, the FPGA 220 of base station 218 has a relatively long amount of computation time to determine if a breakpoint event has occurred or not. In the event a breakpoint has occurred, the microcontroller 232 can be halted and the host processor 210 is informed.

An advantage of this process is that the FPGA 220 and the microcontroller 232 can be stopped at the same time in response to a breakpoint event. Another advantage is that complex and robust breakpoint events are allowed while still maintaining breakpoint synchronization between the two devices. These advantages are achieved with minimal specialized debugging logic (to send I/O data over the interface) and without special bond-out circuitry being required in the microcontroller device under test 232.

A transfer of I/O data as previously described with reference to Figures 3-4 is illustrated with reference to the timing diagram of FIGURE 5. After the

microcontroller 232 completes an I/O read instruction, it sends the read data back to the base station 218 to the virtual microcontroller, since the virtual microcontroller 220 of the present embodiment implements only the core processor functions (and not the I/O functions). The ICE system can expect the incoming data stream for an I/O read to commence with the first positive edge of U_HCLK (the debug or system clock) when SOI signal for the following instruction is at a predetermined logic level (e.g., a logic high). Thus, at time T1, the SOI signal makes a transition to a logic high and one system clock cycle later at time T2, the data transfer phase 310 begins. This timing allows the ICE system to get the read data to the emulated accumulator of base station 218 before it is needed by the next instructions execution. Note that the first SOI pulse shown in FIGURE 5 represents the first SOI following the I/O read instruction (but could be any suitable reference time signal). Transfer of the data from the microcontroller 232 is carried out using the two data lines (data2 and data1, shown as U_D0_BRK and U_D1_IRQ) with each line carrying four bits of an eight bit word. During this data transfer phase 310, an eight bit transfer representing the I/O read data can take place from the microcontroller 232 to the base station 210 in the four clock cycles between T2 and T3. The control phase 318 starts at time T3 and continues until the beginning of the next data transfer phase 310. The SOI signal at T4 indicates that the next data transfer phase is about to start and serves as a reference time to read the data2 line to detect the presence of any halt signal from the virtual microcontroller 220. The current control phase 318 ends at T5 and the next data transfer phase 310 begins.

The base station 218 only transmits break (halt) commands to the microcontroller 232 during the control phase. After the microcontroller 232 is halted in response to the break command, the interface can be used to implement memory / register read / write commands. The halt command is read at the SOI signal transition (T1 or T4). The microcontroller 232 uses the interface to return register information when halted, and to send I/O read, interrupt vector and watchdog timer information while running.

In the case of an interrupt, if an interrupt request is pending for the microcontroller 232, the system asserts U_D1_IRQ as an interrupt request during the control phase of the microcontroller 232. Since the interrupt signal comes to the virtual microcontroller 220 from the microcontroller 232 during the control phase, the virtual microcontroller 220 knows the timing of the interrupt signal going forward. That is, the interrupt signal is the synchronizing event rather than the SOI signal. In case of an interrupt, there is no SOI, because the microcontroller 232 performs special interrupt processing including reading the current interrupt vector from the interrupt controller. Since program instructions are not being executed during the interrupt processing, there is no data / control phase. The virtual microcontroller 220 expects the interrupt vector to be passed at a deterministic time across the interface during this special interrupt processing and before execution of instructions proceeds. Since the virtual microcontroller 220 of the current embodiment does not implement an interrupt

controller, interrupt vectors are read from the interrupt controller upon receipt of an interrupt request over the interface. The interrupt vector data is passed over the interface using the two data lines as with the I/O read data, following the assertion of an internal microcontroller IVR_N (active low) signal during the control phase. In the current embodiment, an interrupt cycle is approximately 10 clock cycles long. Since the interrupt service cycle is much longer than the time required to transfer the current interrupt vector, the data is easily transferred using the two data lines, with no particular timing issues.

If the microcontroller 232 undergoes a watchdog reset, it asserts the IRQ (interrupt) and BRQ (break) lines indefinitely. The ICE detects this condition and further detects that the microcontroller clock has stopped. This is enough to establish that a watchdog reset has occurred. The ICE applies an external reset, and notifies the ICE software in the host computer 210.

METHOD FOR PERFORMING A SLEEP OPERATION

Referring now to Figure 6, a method for performing a sleep operation is disclosed for a system that includes a device under test and that includes an emulator device that emulates the functions of the device under test. During operation of the device under test and the emulator device, a sequence of instructions are executed by the device under test as shown by step 601. The device under test conveys clock signals to the emulator device as shown by step 602. The emulator device executes the sequence of instructions in lock-step

fashion as shown by step 603. In the present embodiment, the device under test is microcontroller 232 of Figure 1 and the emulator device is virtual microcontroller (FPGA) 220 of Figures 1-2 that operate in lock-step fashion as shown in Figures 3-5.

When the sleep function is to be initiated, the operating program sends a first signal to the device under test. In one embodiment, the first signal is generated internally within microcontroller 232 of Figure 1 and the first signal is sent to a register within microcontroller 232 that is used for initiating the sleep function. However, alternatively, other sources could generate the first signal.

As shown by steps 604-605, upon receiving the first signal at the device under test, the sleep function is initiated by the device under test. In the current embodiment the sleep function is a standard function performed by microcontroller 232 in which execution of instructions are halted. The sleep function is commonly used to conserve power.

As shown by step 606 the clocks are turned off. In the embodiment shown in Figures 1-2, microcontroller 232 is operable upon initiating the sleep function to shut down the clocks. In the present embodiment microcontroller 232 is operable to shut down the microcontroller CPU clock and the data clock.

Referring to step 607, upon turning off the clocks as shown in step 606, the emulator device discontinues execution of the sequence of instructions. More particularly, with reference to Figure 1, FPGA 220 ceases execution of the sequence of instructions that are performed in lock-step with microcontroller 232 when clock signals are no longer received. In the present invention, FPGA ceases performing the core processing functions of microcontroller 232 (which are performed in lock-step). However, other communications and operations can still be conducted such as, for example, communication with host computer 210 shown in Figures 1-2.

As shown by steps 608-609 when the device under test has completed the sleep function, the device under test turns back on the clocks (step 609) and sends a second signal, referred to as a "wake-up signal" to the emulator device (step 610). This wake-up signal is simply an indication that the emulator device is to wake up. With reference to Figure 1, in the present embodiment, the wake-up signal is a pulse on interface 242.

As shown by step 611, a determination is made as to the number of clock signals received at the emulator device since the wake-up signal was received. With reference to Figure 1, in the present embodiment, FPGA 220 initiates a counter that counts the number of clock signals received.

The execution of instructions at the emulator device is resumed as shown by steps 612-613 when the determined number of clock signals received at the emulator device since the wake-up signal was received equals a predetermined value. More particularly, with reference to Figure 1, in the present embodiment, FPGA 220 resumes execution of the set of instructions that are performed in lock-step with microcontroller 232 (e.g., the core processing functions of microcontroller 232).

In the present embodiment, upon receiving the signal sent in step 610 a counter is reset. The counter is incremented each time that a clock signal is received. When the number of received clock signals equals the predetermined value, execution of code resumes. In the present embodiment, a predetermined value of seven is used. However, other values could also be used. In the present embodiment, microcontroller 232 resumes execution of instructions after the predetermined number of clock signals have been sent. Thereby, both the device under test and the emulator device resume execution of code in lock-step fashion.

METHOD FOR PERFORMING A STALL OPERATION

Referring now to Figure 7, a method 700 for performing a stall operation is disclosed for a system that includes a device under test and that includes an emulator device that emulates the functions of the device under test. During operation of the device under test and the emulator device, instructions are

executed by the device under test as shown by step 701. The device under test conveys clock signals to the emulator device as shown by step 702. The emulator device executes the instructions in lock-step fashion as shown by step 703. In the present embodiment, the device under test is microcontroller 232 of Figure 1 and the emulator device is virtual microcontroller (FPGA) 220 of Figures 1-2 that operate in lock-step fashion as shown in Figures 3-5.

When the stall function is to be initiated, the operating program sends a first signal to the device under test. As shown by steps 704-705, upon receiving the first signal at the device under test, the stall function is initiated by the device under test. In one embodiment, the first signal is generated internally within microcontroller 232 of Figure 1 and the first signal is sent to a register within microcontroller 232 that is used for initiating the stall function. However, alternatively, other sources could generate the first signal.

As shown by step 706 the device under test discontinues sending clock signals to the emulator device. In the embodiment shown in Figures 1-2, microcontroller 232 is operable upon initiating the stall function to bring the microcontroller clock signal transmitted over the second clock interface line to low and maintain a low signal. Thereby, clock signals are no longer sent to virtual microcontroller (FPGA) 220. However, in the present embodiment, the microcontroller CPU clock continues to run as the stall function is carried out by microcontroller 232.

Referring to step 707, the emulator device discontinues execution of instructions. More particularly, with reference to Figure 1, in the present embodiment, FPGA 220 ceases execution of the set of instructions that are performed in lock-step with microcontroller 232 when clock signals are no longer received over the second clock interface line. In the present invention, the FPGA ceases performing the core processing functions of microcontroller 232 (which are performed in lock-step). However, other communications and operations can still be conducted such as, for example, communication with host computer 210 shown in Figures 1-2.

As shown by steps 708-709 when the stall function has been completed by the device under test, the device under test resumes sending clock signals to the emulator device. In the embodiment shown in Figures 1-2, microcontroller 232 is operable when the stall function is completed to resume operation of the microcontroller CPU clock. Thereby, clock signals are sent from microcontroller 232 to virtual microcontroller (FPGA) 220.

The emulator device is operable upon receiving the clock signals to resume execution of the instructions as shown by step 710. More particularly, with reference to Figure 1, in the present embodiment, FPGA 220 resumes execution of the set of instructions that are performed in lock-step with microcontroller 232 (e.g., the core processing functions of microcontroller 232)

upon receiving the clock signals. In the present embodiment, execution of instructions that are to be performed in lock-step resumes upon the first received clock signal. However, alternatively execution of instructions that are performed in lock-step can be resumed after a predetermined number of clock cycles.

The device under test also resumes execution of instructions upon resumption of generation of clock signals. More particularly, in the embodiment shown in Figures 1-2, microcontroller 232 is operable when the stall function is completed to resume operation of microcontroller CPU clock. Microcontroller 232 then immediately begins to execute instructions. Accordingly, both microcontroller 232 and FPGA device 220 resume execution of instructions immediately upon resumption of operation of the CPU clock of microcontroller 232. Thereby, both the device under test and the emulator device resume execution of instructions in lock-step fashion.

Although the embodiments of the current invention have been explained in terms of providing in-circuit emulation of the core processing functions of a microcontroller, the present invention can be realized for any complex electronic device for which in-circuit emulation is needed including, but not limited to, microprocessors and other complex large scale integration devices without limitation. Moreover, although the mechanism for use of the interface between the host processor and the FPGA has been described in the environment of an ICE system, this should not be considered limiting since this interface

mechanism can be used for other systems requiring FPGA programming and communication functions over a single interface.

Those skilled in the art will recognize that the present invention has been described in terms of exemplary embodiments based upon use of a programmed processor. However, the invention should not be so limited, since the present invention could be implemented using hardware component equivalents such as special purpose hardware and/or dedicated processors which are equivalents to the invention as described and claimed. Similarly, general purpose computers, microprocessor based computers, micro-controllers, optical computers, analog computers, dedicated processors and/or dedicated hard wired logic may be used to construct alternative equivalent embodiments of the present invention.

Those skilled in the art will appreciate that the program steps and associated data used to implement the embodiments described above can be implemented using disc storage as well as other forms of storage such as for example Read Only Memory (ROM) devices, Random Access Memory (RAM) devices; optical storage elements, magnetic storage elements, magneto-optical storage elements, flash memory, core memory and/or other equivalent storage technologies without departing from the present invention. Such alternative storage devices should be considered equivalents.

The present invention, as described in embodiments herein, is implemented using a programmed processor executing programming instructions that are broadly described above in flow chart form that can be stored on any suitable electronic storage medium or transmitted over any suitable electronic communication medium. However, those skilled in the art will appreciate that the processes described above can be implemented in any number of variations and in many suitable programming languages without departing from the present invention. For example, the order of certain operations carried out can often be varied, additional operations can be added or operations can be deleted without departing from the invention. Such variations are contemplated and considered equivalent.

While the invention has been described in conjunction with specific embodiments, it is evident that many alternatives, modifications, permutations and variations will become apparent to those skilled in the art in light of the foregoing description. Accordingly, it is intended that the present invention embrace all such alternatives, modifications and variations as fall within the scope of the appended claims.